towardsdatascience.com /intuitions-on-l1-and-l2-regularisation-235f2db4c261

Intuitions on L1 and L2 Regularisation - Towards Data Science

Raimi Karim ⁝ 9-11 minutes ⁝ 10/5/2020

Photo by rawpixel on Unsplash

Explaining how L1 and L2 work using gradient descent

(Jump right here to skip the introductions.)

Changelog:
27 Mar 2020: Added absolute to the terms in 2-norm and p-norm. Thanks to for pointing this out.

O**verfitting** is a phenomenon that occurs when a machine learning or statistics model is tailored to a particular dataset and is unable to generalise to other datasets. This usually happens in complex models, like deep neural networks.

Regularisation is a process of introducing additional information in order to prevent overfitting. The focus for this article is L1 and L2 regularisation.

There are many explanations out there but honestly, they are a little too abstract, and I'd probably forget them and end up visiting these pages, only to forget again. In this article, I will be sharing with you some intuitions why L1 and L2 work by explaining using **gradient descent**. Gradient descent is simply a method to find the 'right' coefficients through iterative updates using the value of the gradient. (This article shows how gradient descent can be used in a simple linear regression.)

Content

3) Gradient Descent
4) How is overfitting prevented?

Let's go!

0) What's L1 and L2?

L1 and L2 regularisation owes its name to L1 and L2 norm of a vector ***w*** respectively. Here's a primer on norms:

$$||\mathbf{w}||_1 = |w_1| + |w_2| + ... + |w_N|$$

1-norm (also known as L1 norm)

$$||\mathbf{w}||_2 = \left(|w_1|^2 + |w_2|^2 + \ldots + |w_N|^2\right)^{\frac{1}{2}}$$

2-norm (also known as L2 norm or Euclidean norm)

$$||\mathbf{w}||_p = \left(|w_1|^p + |w_2|^p + \ldots + |w_N|^p\right)^{\frac{1}{p}}$$

p-norm

<change log: missed out taking the absolutes for 2-norm and p-norm>

A linear regression model that implements L1 norm for regularisation is called **lasso regression**, and one that implements (squared) L2 norm for regularisation is called **ridge regression**. To implement these two, note that the linear regression model stays the same:

$$\hat{y} = w_1x_1 + w_2x_2 + ... + w_Nx_N + b$$

but it is the calculation of the loss function that includes these regularisation terms:

$$Loss = Error(y, \hat{y})$$

Loss function with no regularisation

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i|$$

Loss function with L1 regularisation

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2$$

Loss function with L2 regularisation

> *Note: Strictly speaking, the last equation (ridge regression) is a loss function with* ***squared*** *L2 norm of the weights (notice the absence of the square root). (Thank you for highlighting this!)*

The regularisation terms are 'constraints' by which an optimisation algorithm must 'adhere to' when minimising the loss function, apart from having to minimise the error between the true *y* and the predicted *ŷ*.

1) Model

Let's define a model to see how L1 and L2 work. For simplicity, we define a simple linear regression model *ŷ* with one independent variable.

$$\hat{y} = wx + b$$

Here I have used the deep learning conventions *w* ('weight') and *b* ('bias').

> *In practice, simple linear regression models are not prone to overfitting. As mentioned in the introduction, deep learning models are more susceptible to such problems due to their model complexity.*

> *As such, do note that the expressions used in this article are easily extended to more complex models, not limited to linear regression.*

2) Loss Functions

To demonstrate the effect of L1 and L2 regularisation, let's fit our linear regression model using 3 different loss functions/objectives:

- L
- L1
- L2

Our objective is to minimise these different losses.

2.1) Loss function with no regularisation

We define the loss function L as the squared error, where error is the difference between *y* (the true value) and *ŷ* (the predicted value).

$$\begin{aligned} L &= (\hat{y} - y)^2 \\ &= (wx + b - y)^2 \end{aligned}$$

Let's assume our model will be overfitted using this loss function.

2.2) Loss function with L1 regularisation

Based on the above loss function, adding an L1 regularisation term to it looks like this:

$$L_1 = (wx + b - y)^2 + \lambda|w|$$

where the regularisation parameter $\lambda > 0$ is manually tuned. Let's call this loss function L1. Note that |*w*| is differentiable everywhere except when *w*=0, as shown below. We will need this later.

$$\frac{d|w|}{dw} = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \end{cases}$$

2.3) Loss function with L2 regularisation

Similarly, adding an L2 regularisation term to L looks like this:

$$L_2 = (wx + b - y)^2 + \lambda w^2$$

where again, $\lambda > 0$.

3) Gradient Descent

Now, let's solve the linear regression model using gradient descent optimisation based on the 3 loss functions defined above. Recall that updating the parameter *w* in gradient descent is as follows:

$$w_{\text{new}} = w - \eta \frac{\partial L}{\partial w}$$

Let’s substitute the last term in the above equation with the gradient of L, L1 and L2 w.r.t. *w.*

L:

$$\begin{aligned} w_{\text{new}} &= w - \eta \frac{\partial L}{\partial w} \\ &= w - \eta \cdot \left[2x(wx + b - y)\right] \end{aligned}$$

L1:

$$\begin{aligned} w_{\text{new}} &= w - \eta \frac{\partial L_1}{\partial w} \\ &= w - \eta \cdot \left[2x(wx + b - y) + \lambda \frac{d|w|}{dw}\right] \\ &= \begin{cases} w - \eta \cdot \left[2x(wx + b - y) + \lambda\right] & w > 0 \\ w - \eta \cdot \left[2x(wx + b - y) - \lambda\right] & w < 0 \end{cases} \end{aligned}$$

L2:

$$\begin{aligned} w_{\text{new}} &= w - \eta \frac{\partial L_2}{\partial w} \\ &= w - \eta \cdot \left[2x(wx + b - y) + 2\lambda w\right] \end{aligned}$$

4) How is overfitting prevented?

From here onwards, let's perform the following substitutions on the equations above (for better readability):

- $\eta = 1$,
- $H = 2x(wx+b-y)$

which give us

L:

$$w_{\text{new}} = w - H \text{ —— } (0)$$

L1:

$$w_{\text{new}} = \begin{cases} (w - H) - \lambda, & w > 0 \quad \text{—— } (1.1) \\ (w - H) + \lambda, & w < 0 \quad \text{—— } (1.2) \end{cases}$$

L2:

$$w_{\text{new}} = (w - H) - 2\lambda w \quad \text{—— } (2)$$

4.1) With vs. Without Regularisation

Observe the differences between the weight updates with the regularisation parameter λ and without it. Here are some intuitions.

Intuition A:

Let's say with Equation 0, calculating *w-H* gives us a *w* value that leads to overfitting. Then, intuitively, Equations {1.1, 1.2 and 2} will reduce the chances of overfitting because introducing λ makes us shift *away* from the very *w* that was going to cause us overfitting problems in the previous sentence.

Intuition B:

Let's say an overfitted model means that we have a *w* value that is **perfect** for our model. 'Perfect' meaning if we substituted the data (*x*) back in the model, our prediction $\hat{y}$ will be very, very close to the true *y*. Sure, it's good, but we don't want perfect. Why? Because this means our model is only meant for the dataset which we trained on. This means our model will produce predictions that are far off from the true value for *other* datasets. So we settle for **less than perfect**, with the hope that our model can also get close predictions with other data. To do this, we 'taint' this perfect *w* in Equation 0 with a penalty term λ*.* This gives us Equations {1.1, 1.2 and 2}.

Intuition C:

Notice that *H* (as defined here) is **dependent** on the model (*w* and *b*) and the data (*x* and *y*). Updating the weights based *only* on the model and data in Equation 0 can lead to overfitting, which leads to poor generalisation. On the other hand, in Equations {1.1, 1.2 and 2}, the final value of *w* is not only influenced by the model and data, but *also* by a predefined parameter *λ* which is **independent** of the model and data. Thus, we can prevent overfitting if we set an appropriate value of *λ*, though too large a value will cause the model to be severely underfitted.

Intuition D:

(thanks!) has provided an intuition about the *direction* toward which our solution is being shifted. Have a look in the comments: https://medium.com/@edden.gerber/thanks-for-the-article-1003ad7478b2

4.2) L1 vs. L2

We shall now focus our attention to L1 and L2, and rewrite Equations {1.1, 1.2 and 2} by rearranging their *λ* and *H* terms as follows:

L1:

$$w_{\text{new}} = \begin{cases} (w - \lambda) - H, & w > 0 \quad \text{———} \quad (3.1) \\ (w + \lambda) - H, & w < 0 \quad \text{———} \quad (3.2) \end{cases}$$

L2:

$$w_{\text{new}} = (w - 2\lambda w) - H \quad \text{———} \quad (4)$$

Compare the second term of each of the equation above. Apart from *H*, the change in *w* depends on the **±*λ* term** or the **-2*λ*w term**, which highlight the influence of the following:

1. sign of current *w* (L1, L2)
2. magnitude of current *w* (L2)
3. doubling of the regularisation parameter (L2)

While weight updates using L1 are influenced by the first point, weight updates from L2 are influenced by all the three points. While I have made this comparison just based on the iterative equation update, please note that this does not mean that one is ‘better’ than the other.

For now, let’s see below how a regularisation effect from L1 can be attained just by the sign of the current *w.*

4.3) L1’s effect on pushing towards 0 (sparsity)

Take a look at L1 in Equation 3.1. If *w* is positive, the regularisation parameter *λ*>0 will push *w* to be less positive, by subtracting *λ* from *w*. Conversely in Equation 3.2, if *w* is negative, *λ* will be added to *w*, pushing it to be less negative. Hence, this has the effect of **pushing *w* towards 0**.

This is of course pointless in a 1-variable linear regression model, but will prove its prowess to ‘remove’ useless variables in multivariate regression models. You can also think of L1 as **reducing the number of features** in the model altogether. Here is an arbitrary example of L1 trying to ‘push’ some variables in a multivariate linear regression model:

$$\hat{y} = 0.4561x_1 - 0.0007x_2 + 0.3251x_3 + 0.0009x_4 + 0.0001x_5 - 0.9142x_6 - 0.553$$

So how does pushing *w* towards 0 help in overfitting in L1 regularisation? As mentioned above, as *w* goes to 0, we are reducing the number of features by reducing the variable importance. In the equation above, we see that *x_2*, *x_4* and *x_5* are almost ‘useless’ because of their small coefficients, hence we can remove them from the equation. This in turn **reduces the model complexity**, making our model simpler. A simpler model can reduce the chances of overfitting.

Note

While L1 has the *influence* of pushing weights towards 0 and L2 does not, this does not imply that weights are not able to reach close to 0 due to L2.

If you find any part of the article confusing, feel free to highlight and leave a response. Additionally, if have any feedback or suggestions how to improve this article, please do leave a comment below!

Special thanks to Yu Xuan, Ren Jie, Daniel and Derek for ideas, suggestions and corrections to this article. Also thank you for pointing out the mistake in the derivative.

Follow me on Twitter @remykarem or LinkedIn. You may also reach out to me via raimi.bkarim@gmail.com. Feel free to visit my website at remykarem.github.io.